Viterbi Algorithm

(10 points)

Finish the implementation of the `ViterbiAlgorithm` class using the dynamic programming approach presented in the lecture. The class has a constructor taking the necessary information of a Hidden Markov Model and a `getStateSequence` method returning the most probable state of sequences for a given sequence of observations.

- The constructor takes the following parameters:
  - `String[] states` - an array containing the set of states Q in the same order as in the transition matrix and the emission matrix.
  - `String[] observationVocab` - an array containing all possible observations V in the order they have in the emission matrix.
  - `double[][] transitionMatrix` - the transition matrix A where `transitionMatrix[i][j]` contains the probability a_{ij} for a transition from q_i to q_j (since the `states` array starts at 0 and not at 1, `transitionMatrix[i][j]` gives the transition from `states[i - 1]` to `states[j - 1]`). Note that the transition matrix has two more states as described in the lecture slides. The start state always has the index `0` marks the start state in the matrix while the final state always has the index `transitionMatrix.length - 1`.
  - `double[][] emissionMatrix` - the emission matrix B containing the probabilities b_{ik} that state q_i emits the observation v_k.
- The `getStateSequence` method takes the following parameters:
  - `String[] observations` - an array containing the observations for which the most probable sequence of states should be returned.
- The method should return an instance of the `StateSequence` class. This class contains two following attributes:
  - `String[] states` - the sequence of states which emitted a certain sequence of observations.
  - `double logProbability` - the logarithm of the probability of the state sequence (including the emission probabilities for the sequence of observations).

Example

The visible tests rely on the ice cream example available in the lecture slides (see lecture slide 14 / page 22 in the PDF). The array of states and observations are given as follows:

```
String[] states = new String[] { "HOT", "COLD" };
String[] observationVocab = new String[] { "1", "2", "3" };
```

double[][] transitionMatrix = new double[][] { { 0, 0.8, 0.2, 0 }, { 0, 0.6, 0.3, 0.1 }, { 0, 0.4, 0.5, 0.1 }, { 0, 0, 0, 0 } };

The transition matrix A taken from the automaton picture looks like the following table. (**Please note** that although the states `HOT` and `COLD` have the ids `0` and `1` in the `states` array, the have the ids `1` and `2` in the transition matrix.)

from \ to	start	HOT	COLD	end
start	0	0.8	0.2	0
HOT	0	0.6	0.3	0.1
COLD	0	0.4	0.5	0.1
end	0	0	0	0

The emission matrix B taken from the automaton picture looks like the following table. (Please note that this matrix does neither contain the start nor the final state since both are not emitting any observation.)

state \ observation	1	2	3
HOT	0.2	0.4	0.4
COLD	0.5	0.4	0.1

Hints

- The input matrices will contain probabilities (not their logarithms)
- The test is separated into two different cells. The first cell will test some example observation sequences and compare it with an expected result. The second cell will do the same but for a larger, generated sequence.
- The number of states and observations are not limited to the ice cream example. For the hidden tests, we will use a different scenario with different observations and states.
- Make sure that the tests take less than 2:00 minutes (the evaluation has a max runtime of 5 minutes per file including the hidden tests)
- Observations **do not have to be** numbers. Please take this into account for your implementation.

Notes

- You are free to use a different IDE to develop your solution. However, you have to copy the solution into this notebook to submit it.
- Do not add additional external libraries.
- Interface
  - You can use *[TAB]* for autocompletion and *[SHIFT]+[TAB]* for code inspection.
  - Use *Menu -> View -> Toggle Line Numbers* for debugging.
  - Check *Menu -> Help -> Keyboard Shortcuts*.
- Known issues
  - All global variables will be set to void after an import.
  - Missing spaces arround `%` (Modulo) can cause unexpected errors so please make sure that you have added spaces around every `%` character.
- Finish
  - Save your solution by clicking on the *disk icon*.
  - Make sure that all necessary imports are listed at the beginning of your cell.
  - Run a final check of your solution by
    - click on *restart the kernel, then re-run the whole notebook* (the fast forward arrow in the tool bar)
    - wait fo the kernel to restart and execute all cells (all executable cells should have numbers in front of them instead of a `[*]`)
    - Check all executed cells for errors. If an exception is thrown, please check your code. Note